Understanding Plug-in Test Suites from an Integration Perspective

Michaela Greiler, Arie van Deursen
Software Engineering Research Group
Delft University of Technology
Understanding Plug-in Test Suites
import junit.framework.TestCase;
import junit.framework.TestSuite;

import org.eclipse.myllyn.common.tests.AllCommonTests;
import org.eclipse.myllyn.content.tests.AllContextTests;
import org.eclipse.myllyn.discovery.tests.AllDiscoveryTests;
import org.eclipse.myllyn.ide.tests.AllIDETests;
import org.eclipse.myllyn.java.tests.AllJavaTests;
import org.eclipse.myllyn.monitor.tests.AllMonitorTests;
import org.eclipse.myllyn.resources.tests.AllResourceTests;
import org.eclipse.myllyn.tasks.tests.AllTasksTests;
import org.eclipse.myllyn.team.tests.AllTeamTests;
import org.eclipse.myllyn.tests.integration.AllIntegrationTests;
import org.eclipse.myllyn.tests.misc.AllMiscTests;

public class AllNonConnectorTests {

    public static Test suite() {
        // the order of these tests might still matter, but shouldn't
        TestSuite suite = new TestSuite("All Non-Connector Tests for org.eclipse.myllyn-tests");
        suite.addTest(AllCommonTests.suite());
        suite.addTest(AllContextTests.suite());
        suite.addTest(AllDiscoveryTests.suite());
        suite.addTest(AllIDETests.suite());
        suite.addTest(AllJavaTests.suite());
        suite.addTest(AllMonitorTests.suite());
        suite.addTest(AllResourceTests.suite());
        suite.addTest(AllTasksTests.suite());
        suite.addTest(AllTeamTests.suite());
        suite.addTest(AllIntegrationTests.suite());
        suite.addTest(AllMiscTests.suite());
        return suite;
    }
}

Bundle-ClassPath: mylyn-tests.jar
Require-Bundle: org.junit,
Need more Information?

Which Information?
Code Dependencies
which plug-ins are using each other

Service Relations
which plug-ins are using services each other

Extension Relations
which plug-ins are extending each other
Plug-in A
<extension-point id="pointID"> 
| 
| implements 
| 
| creates, calls 
| 
| contributes to 
| 
| Plug-in B
<extension name="extensionB" point="A.pointID"> 
| 
| class ExtensionB 
| 
| 
| 
| 

Show View

- General
- Ant
- API Tooling
- CVS
- Debug
- ETSE View Category
- Help
- Java
- Java Browsing
- Plug-in Development
- PlugIn Category
- PlugIn View
- Tasks
- Team
- WindowBuilder
Interviewed 25 Eclipse practitioners

Understanding test cases

Understanding test suites
Interviewed 25 Eclipse practitioners

Coding standards, naming conventions.

Explanations and Motivation.

“Tests are also like other parts of code. Sometimes people give bad names to their methods and variables. Then it is hard to understand.”
Interviewed 25 Eclipse practitioners

“What one specific test does, that’s quite easy to understand. What’s difficult is to see where the blank spots on the map are.”
Interviewed 25 Eclipse practitioners

Challenges keeping an Overview:

“It is difficult. You have to read it all. If you skip one test, you do not know one part of the system.”

Test Organization and Structure.

Nested Test Suites.
Interviewed 25 Eclipse practitioners

“We realized: either we work on a particular point, and we run only one test case, or we run all of them.

We worry very often about not being able to run the test suite in a more fine-grained way..”
public class AllNonConnectorTests {

    public static Test suite() {
        // the order of these tests might still matter, but shouldn't
        Testsuite suite = new Testsuite("All Non-Connector Tests for org.eclipse.mylyn.tests");
        suite.addTest(AllCommonTests.suite());
        suite.addTest(AllDiscoveryTests.suite());
        suite.addTest(AllIdTests.suite());
        suite.addTest(AllJavaTests.suite());
        suite.addTest(AllMonitorTests.suite());
        suite.addTest(AllResourceTests.suite());
        suite.addTest(AllTeamTests.suite());
        suite.addTest(AllIntegrationTests.suite());
        suite.addTest(AllTeamTests.suite());
    }
}

Import junit.framework.Test;
Import junit.framework.TestSuite;
Import org.eclipse.mylyn.common.tests.AllCommonTests;
Import org.eclipse.mylyn.content.tests.AllContentTests;
Import org.eclipse.mylyn.discovery.tests.AllDiscoveryTests;
Import org.eclipse.mylyn.ide.tests.AllIdTests;
Import org.eclipse.mylyn.java.tests.AllJavaTests;
Import org.eclipse.mylyn.monitor.tests.AllMonitorTests;
Import org.eclipse.mylyn.resources.tests.AllResourceTests;
Import org.eclipse.mylyn.team.tests.AllTeamTests;
Import org.eclipse.mylyn.tests.integration.AllIntegrationTests;
Import org.eclipse.mylyn.tests.misc.AllMiscTests;
Which plug-ins are tested by which test-component?

How are the plug-in extensions and services tested?

Which extensions are used during the tests?

How are the plug-ins extending each other?
Manual Inspection!
Look at...

...extension relations

Only Static

tests

dependencies
Understanding Plug-in Test Suites from an Integration Perspective
## 5 Architectural View

### Plug-in Modularization View
- **Plug-in Level**
  - Plug-in A uses Plug-in B

### Test-Suite Modularization View
- **Test Case Level**
  - Plug-in A is tested by test case $t$ of Plug-in B

### Extension Initialization View
- **Extension Level**
  - Plug-in A loads extension $e$ of Plug-in B

### Extension Usage View
- **Method Level**
  - Plug-in A invokes method $m$ of extension $e$ in plug-in B

### Service Usage View
- **Method Level**
  - Plug-in A invokes method $m$ of service $s$ in plug-in B
Ingredients: Data Behind

Static Data & Dynamic Trace Data

Fact Extraction
- plugin.xml & schema
- manifest
- class files

Instrumentation
- Test suite
- class files

Reconstruction

VIEWS

Recipe: Reconstructing The Views
Recipe: Reconstructing the Extension Usage View

**Fact Extraction**
- STATIC: plug-in.xml, schema, .class

**Instrumentation**
- DYNAMIC: Method Calls

**Reconstruction**
- Extension Method Set: Heuristic (all methods of an extension)
- Inspect the Trace File

**Extension Usage View**
- which “extension methods” are invoked
ETSE: Eclipse Test Suite Explorer
Empirical Evaluation
Case Study Set-Up

- **eclipse Mylyn**
  - 27 plug-ins
  - 11 test-components
  - 2 test suites
  - 200,000 LOCs
  - 30,000 LOCs tests

- **EGit**
  - 5 main plug-ins
  - 2 test components
  - 4 test suites
  - 100,000 LOCs
  - 14,000 lines tests

- **trac Connector**
  - 3 main plug-ins
  - 1 test component
  - 2 test suites
  - 8,500 LOCs
  - 3,400 lines tests
Publications

Understanding Plug-in Test Suites from an Extensibility Perspective
Michaela Greiler, Hans-Gerhard Gross, Arie van Deursen
Working Conference on Reverse Engineering (WCRE), October 13-17 2010, Boston, USA
Download: PDF

Abstract:
Plug-in architectures enable developers to build extensible software products. Such products include plug-ins, and their functionality can be enriched by adding or configuring plug-ins. The platform also of multiple plug-ins, and offer dedicated points through which their functionality can be accessed. A well-known example of such an architecture is Eclipse, best known for its use to create a series
ETSE: Eclipse Test Suite Explorer
Which plug-ins are tested by which test-component?

```
mylyn.commons.ui
mylyn.commons.net
mylyn.commons.core
mylyn.commons.xmlrpc
```

Static

Dynamic
How are the plug-ins extending each other?

How are the plug-in extension-relations tested?

System Level
How are the plug-ins extending each other?

How are the plug-in extension-relations tested?

Plug-in Level

Dynamic & Static
Which extensions are used during the tests?

When and how have the extensions been used?

<table>
<thead>
<tr>
<th>Used Extensions</th>
<th>Involved Test Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Extension</strong></td>
<td><strong>Extension Point</strong></td>
</tr>
<tr>
<td>1. org.eclipse.mylyn.context.ui</td>
<td>org.eclipse.ui.viewActions</td>
</tr>
<tr>
<td>2. org.eclipse.mylyn.context.ui</td>
<td>org.eclipse.mylyn.tasks.ui.editors</td>
</tr>
<tr>
<td>5. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.mylyn.tasks.ui.presentations</td>
</tr>
<tr>
<td>6. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.ui.views</td>
</tr>
<tr>
<td>7. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.ui.actionSets</td>
</tr>
<tr>
<td>8. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.ui.editors</td>
</tr>
<tr>
<td>9. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.mylyn.tasks.ui.editors</td>
</tr>
<tr>
<td>10. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.mylyn.tasks.ui.reposiories</td>
</tr>
<tr>
<td>11. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.mylyn.tasks.ui.projectlinkProviders</td>
</tr>
<tr>
<td>12. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.ui.exportWizards</td>
</tr>
<tr>
<td>13. org.eclipse.mylyn.tasks.ui</td>
<td>org.eclipse.ui.importWizards</td>
</tr>
</tbody>
</table>
Want to use ETSE in your project?

Contact us!

m.s.greiler@tudelft.nl
arie.vandeursen@tudelft.nl

Update site:
http://swerl.tudelft.nl/etse/ETSE_UpdateSite/site.xml